19. Функции

Проф. д-р Емил Хаджиколев

1. Механизми за присвояване по стойност и по адрес
2. Предаване на параметри по стойност и по адрес
3. Връщана стойност

Присвояване по стойност

- При присвояването по стойност, на една променлива присвояваме друга.
- При това, стойността на втората се копира като стойност на първата.

```
int i = 1;
int j = i; // присвояване по стойност
j = 5;     // i не се променя
```

- След присвояването, двете променливи стават независими една от друга – имат самостоятелни памети за съхраняване на стойностите.
- В C++ по подразбиране обикновените присвоявания (на променливи от примитивни или съставни типове) са по стойност.

Присвояване по адрес

- При присвояването по адрес, две различни променливи сочат (работят с) обща памет.
- Променяйки стойността чрез едната променлива, се променя и стойността на другата.
- В C++ има два механизма за присвояване по адрес
  - псевдоними – (най-вече) за статични обекти;

```
int i = 1;
int &j = i; // присвояване по адрес чрез псевдоним
j = 5;      // i се променя
```

  - указатели – за статични и динамични обекти;

```
int i = 1;
int *j = &i; // присвояване по адрес чрез указател
*j = 5;      // i се променя
```

Предаване на параметри към функции по стойност и адрес

- Механизмите на предаване на параметри не се различават от присвояването.
- При извикване на функция, на формалните ѝ параметри се присвояват фактическите.
- Правилата за декларация на формални параметри като псевдоними и указатели са същите като при присвояването.
- Ако в тялото на функция се променят формалните параметри предадени по адрес, то ще се променят и съответните фактически параметри.

Пример: предаване по адрес и по стойност (1)

```
// Предаване по стойност
void m1(int i){
        i = 3;
}
// Предаване по адрес чрез псевдоним
void m2(int &i){ // с & указваме че i е псевдоним
        i = 3;   // с псевдоним се работи като с обикновена променлива
}
// Предаване по адрес чрез указател
void m3(int *i){ // с * казваме, че i е указател
        *i = 3;   // стойността на указател се взима с оператора * -
                  // *i е стойността, към която сочи указателя i
}
```

Пример: предаване по адрес и по стойност (2)

```
int main() {
   int i=5, j=5, k=5;

   m1(i);  // предаване по стойност
   cout << i << '\n'; // 5

   m2(j);  // предаване по адрес чрез псевдоним
   cout << j << '\n'; // 3

   m3(&k); // предаване на адрес към указател
   cout << k << '\n'; // 3

   return 0;
}
```

Разлики при предаване по адрес и по стойност

- При предаването по стойност, при всяко извикване на подпрограма, в стека (може и в динамичната памет за полета) се заделя памет за формалния параметър.
- Предаването по адрес спестява заделянето на памет и се изпълнява по-бързо.
- Може да предаваме параметър по адрес или по стойност и да забраним промяната му в тялото на функцията, като при декларация на формалния параметър се използва ключовата дума const след типа:

```
void m2(int const &i)
void m3(int const *i)
```

Страничен ефект

- Промяната на фактически параметър при извикване на подпрограма се нарича страничен ефект.
- Страничните ефекти обикновено са лоша техника за програмиране и се избягват (но при работа с масиви може да се използват).
- Вместо използване на страничен ефект, се предпочита функцията да връща стойност като резултат от изпълнението си.

Масиви и указатели

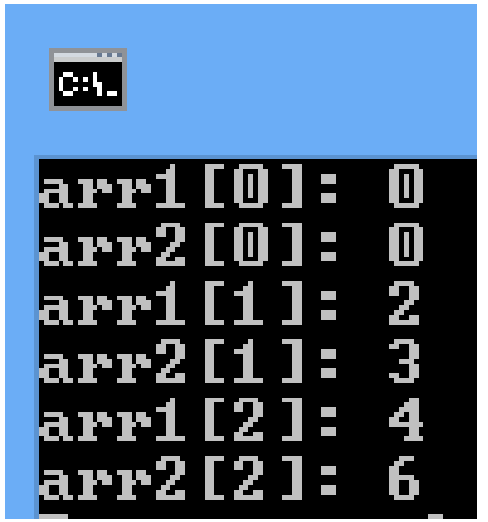
- Имената на масивите са указатели, без значение как са дефинирани сочените от тях масивите – като статични или динамични.
- Съответно, със статични или динамични масиви се работи еднотипно.
- Ако броят не елементите на масива не е известен при създаване на програмата, може да използваме динамични масиви – при които броят на елементите може да се определя по време на изпълнение на програмата.
- При статичните масиви броят на елементите е константа.

Масиви и указатели - пример

```
int n = 3;
int arr1[3];              // статичен масив
int *arr2 = new int[n];   // динамичен масив

int *refInt = new int;    // указател към една стойност от тип int
                          // декларациите на динамичен масив и обект си приличат
for (int i = 0; i<n; i++)
{
     arr1[i] = i * 2;
     arr2[i] = i * 3;
     cout << "arr1[" << i << "]: " << arr1[i] << '\n';
     cout << "arr2[" << i << "]: " << arr2[i] << '\n';
}

delete []arr2;
delete refInt;    // освобождаване на паметта за динамичните обекти
```



Предаване на масиви като параметри

- Масивите се предават винаги по адрес – защото имената са указатели.
- Формалният параметър за масив може да се дефинира като масив или като указател без значение как е създаден масива (статичен или динамичен):
  - int array[]
  - int *array
- В тялото може да работим еднотипно с масивите.

Предаване на масиви като параметри - пример

```
//void array_info(int *data, int size)
void array_info(int data[], int size) {
    for (int i = 0; i < size; i++) {
        cout << data[i] << (i<size-1?", ":"");
    }
    cout << '\n';
}
int main() {
    const int size = 4;
    int list[size] = { 3, 6, 6, 8 };

    array_info(list, size);

    return 0;
}
```

Адресна аритметика. Оператори за работа с указатели

- Стойността на променлива-указател е адрес от паметта – някакво число.
- Указател сочи към стойност от конкретен тип:

```
int *i; double *d;
```

- Към променлива-указател може да добавяме или изваждаме цяло число – n:

```
i + 1; d + 2;
```

- При това към адреса се добавя/изважда число равно на броя байтове за съответния тип, умножени по указаното събираемо:

  към i се добавят 4 байта*n (в примера 4), към d – 8*n (в примера 16).

- В резултат се получава адрес на друга клетка в паметта.
- За да има смисъл от подобна работа трябва да очакваме, че на новия адрес се намират данни от типа на указателя. Това може да се използва при работа с масиви.
- **Операциите събиране и изваждане на указател с число се наричат адресна аритметика.**

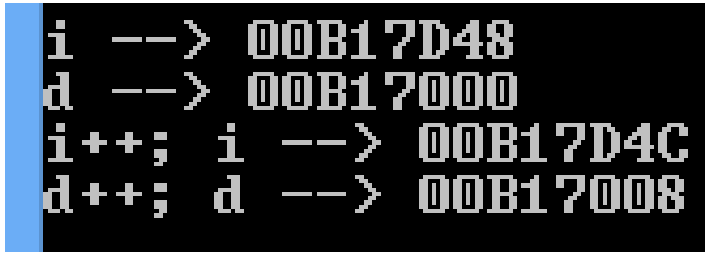
Адресна аритметика. Оператори за работа с указатели – пример

```
int *i = new int;         // i е указател към int
double *d = new double; // d е указател към double

cout << "i --> " << i << endl; // извежда се адреса на i
cout << "d -->" << d << endl; // извежда се адреса на d

i++; // към адреса на i добавяме 1 = 1*4 (байта)
d++; // към адреса на d добавяме 1 = 1*8 (байта)

cout << "i++; i --> " << i << endl; // извежда новия адрес на i
cout << "d++; d --> " << d << endl; // извежда новия адрес на d
```



Адресна аритметика и масиви

- Променливата за масив е константен указател, съхраняващ адреса на първия елемент (с индекс 0) от масива.
- Елементите на масив се съхраняват в последователни клетки от паметта.
- За работа с масиви може да се използва адресна аритметика:
  - **указател към i-ти елемент** в масива arr: **arr+i**
  - **стойност на i-ти елемент**: ***(arr+i)**
    - стойност на елемент с индекс 0 – arr[0] <--> *(arr+0) <--> *arr
    - стойност на елемент с индекс 1 – arr[1] <--> *(arr+1)
    - стойност на елемент с индекс 2 – arr[2] <--> *(arr+2)

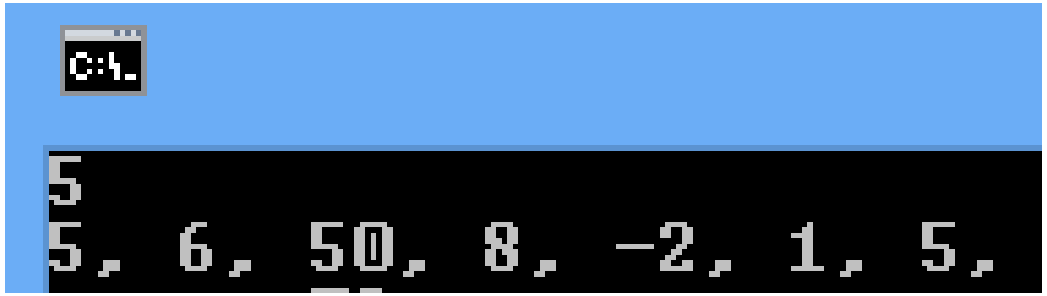
Адресна аритметика и масиви - пример

```
const int size = 7;
int arr[size] = {5, 6, -3, 8, -2, 1, 5 };

cout << *arr << '\n';
// Обръщение към първия елемент в масива arr[0]

*(arr+2) = 50;
// Задаване на нова стойност на 3-тия елемент arr[2]

for (int i = 0; i<size; i++) {
      cout << *(arr + i) << ", ";
}
```



Връщана стойност на функция

- Като тип на връщана стойност на функция може да се опише всякакъв тип:
  - примитивен
  - съставен
  - указател
  - псевдоним

Връщана стойност на функция - примери

```
int* getInt(){
   int *i = new int(4);
   return i;
}
```

```
int& getInt2(){
   int i = 4;
   return i;
}
```

```
Point2D* getPoint(){
   Point2D *p = new Point2D;
   p->x = 5; p->y = 6;
   return p;
}
int main() {
   int *i = getInt();
   int j = getInt2();
   Point2D *p = getPoint();
   return 0;
}
```